Universiteit Gent - TELIN - DDCM
2024-2025

Workshop: DML 1

1 Introductie

In de vorige workshops heb je geleerd hoe je met een relationele PostgreSQL databank kan communiceren en hoe je de basiscomponenten van een relationele databank fysiek kan implementeren. In deze workshop gaan we dieper in op het toevoegen van data aan een relationele databank. Tijdens deze workshop en de workshop 'DML 2' zullen we SQL-instructies introduceren die gecategoriseerd worden onder de 'Data Manipulation Language' (DML) van SQL. Hieronder vallen alle intructies waarmee je data kan manipuleren (toevoegen, aanpassen, verwijderen). Opnieuw kiezen we voor het PostgreSQL databankbeheersysteem. Ook zullen we, net als in de vorige workshops, werken met de `wielrennen` databank. Voor de volledigheid is het relationele databankschema van de `wielrennen` databank terug te vinden in de Appendix van deze workshop. Ook vinden jullie op Ufora (in de map van de workshop 'DDL 1') een SQL-script dat alle instructies bevat voor een eerste fysieke implementatie van deze databank. Zo kunnen jullie met een correcte databankimplementatie starten aan deze workshop, hoewel je meer zal leren als je deze workshop maakt vertrekkende van je eigen oplossing.

2 Eerste stappen

Onderstaande sectie dien je enkel door te nemen indien je nog niet (volledig) klaar bent met de vorige workshops of indien je wil vertrekken van onze modeloplossing. Indien je hier langer dan 15 minuten aan spendeert, vraag dan hulp aan een van de assistenten.

Vooraleer we effectief gaan leren hoe je data kan toevoegen aan de `wielrennen` databank, komen we eerst nog even kort terug op een aantal zaken uit de vorige workshops.
Ten eerste willen we jullie er aan herinneren dat alle informatie over het downloaden en installeren van PostgreSQL en alle gerelateerde tools staat opgelijst in een handleiding op Ufora. Vooraleer jullie starten met deze workshop is het aangeraden om deze handleiding door te nemen, en PostgreSQL en pgAdmin 4 te installeren indien je dit nog niet eerder deed.
Daarnaast wordt er verwacht dat jullie de vorige workshops ('Introductie PostgreSQL' en 'DDL 1') volledig hebben afgewerkt. Idealiter hebben jullie na het afwerken van deze workshops een `wielrennen` databank waarin alle basiscomponenten fysiek geïmplementeerd zijn en/of een backup in de vorm van een SQL-script van deze databank. Mocht dit niet het geval zijn, dan kunnen jullie een volledig backup-script (met naam `wielrennen_ddl1.sql`) terugvinden in de map van de workshop 'DDL 1' op Ufora. Dit script kan je gebruiken als voorbeeldoplossing van het (basis) fysiek ontwerp van de `wielrennen` databank. Indien jullie dit script wensen te gebruiken, is het aangeraden om de oplossing in detail door te nemen vooraleer verder te gaan en nog eens na te gaan of je alle concepten uit de vorige workshops goed begrepen hebt.
Een fysieke implementatie van de `wielrennen` databank bekomen jullie door deze backup in te laden (restore). Dit kan je doen via de commandolijn-applicatie `psql` met het volgende commando (plaats het volledige commando op 1 lijn).

```
psql
    --host=127.0.0.1
    --port=5432
    --dbname=wielrennen
    --username=postgres
    --file=wielrennen_ddl1.sql
```

Let wel op dat er reeds een lege `wielrennen` databank moet bestaan op je lokale PostgreSQL-cluster. Deze dien je dus eerst zelf aan te maken.

> Zorg, vooraleer verder te gaan, dat je een volledig geïmplementeerde `wielrennen` databank hebt aangemaakt op je lokale PostgreSQL-cluster.

3 Toevoegen

Wanneer een databank ontworpen en geïmplementeerd is, kan ze opgevuld worden met data. Typisch kunnen data op drie verschillende manieren worden toegevoegd

aan bestaande basisrelaties. Ten eerste is het mogelijk om waarden in te laden die opgeslagen zijn in een extern databestand (zie Sectie 3.1). Daarnaast kan data uit een basisrelatie gekopieerd worden naar een andere basisrelatie door middel van (eenvoudige) `SELECT`-queries (zie Sectie 3.2). Tenslotte is het ook mogelijk om expliciet waarden voor een of meerdere rijen op te geven en deze rijen toe te voegen aan een basisrelatie (zie Sectie 3.3). Deze drie varianten voor het toevoegen van data worden hieronder in detail besproken.

3.1 Data importeren uit een bestand

Een eerste methode om data toe te voegen aan een bestaande basisrelatie is door data die opgeslagen zijn in een extern databestand in te laden (of te 'importeren') in deze basisrelatie. Zo'n databestand kan bijvoorbeeld gegenereerd zijn door een applicatie of kan overgedragen zijn tussen verschillende bedrijven of personen. Het is zelfs mogelijk dat een databestand aangemaakt is door rechtstreeks data te exporteren uit een (andere) databank.
In deze workshop concentreren we ons op het inladen van data die zich in een .csv bestand bevinden in een basisrelatie die geïmplementeerd is in een PostgreSQL databank. De extensie '.csv' staat voor 'comma-separated values' en de structuur van zo'n bestand komt sterk overeen met een tabel, zoals we die reeds kennen uit het relationele model. Meerbepaald betekent dit dat elke rij in een .csv bestand overeenkomt met een data-object (een entiteit), elke kolom overeenkomt met een attribuut van het data-object, en de kolomwaarden typisch gescheiden worden door een komma. Er zijn echter ook andere scheidingstekens mogelijk, zoals de puntkomma of de tab. Vaak bevat de eerste lijn van een .csv bestand een oplijsting van alle kolomnamen. Dit wordt de header genoemd en het is belangrijk dat deze header niet als data geïmporteerd wordt in een basisrelatie.
Voor de `wielrennen` databank hebben we twee .csv bestanden (met bestandsnaam `renners.csv` en `uitslagen.csv`) voorzien waarin zich alle (voorbeeld)data bevinden. Deze bestanden zijn ook terug te vinden op Ufora in de map van deze workshop. Het bestand met naam `renners.csv` bevat data met betrekking tot wielrenners en wielerteams. Daarnaast bevat het bestand met naam `uitslagen.csv` data met betrekking tot wedstrijden, routes en uitslagen.

> Open de bestanden `renners.csv` en `uitslagen.csv` met een tekstverwerker (bv. Notepad++, niet met Excel...) en bekijk aandachtig de inhoud.

Wanneer je deze bestanden bekijkt, zal je (hopelijk) vaststellen dat geen enkele basisrelatie die je tijdens de workshop 'DDL 1' hebt aangemaakt dezelfde structuur

vertoont. Het is dus ook niet mogelijk om de data die zich in deze bestanden bevinden rechtstreeks in te laden in een bestaande basisrelatie. Wel stellen wij volgend stappenplan voor dat je steeds kan volgen als je data die in .csv bestanden zijn opgeslagen wil inladen in een relationele databank. Deze stappen worden verder nog in detail besproken.

1. Maak voor elk .csv-bestand een nieuwe 'super'-relatie aan in de `wielrennen` databank. Dit is eigenlijk niet meer dan een gewone basisrelatie die bestaat uit *alle* attributen die voorkomen in het overeenkomstige .csv bestand. Aangezien alle data in een .csv bestand typisch tekstueel zijn, is het het eenvoudigste om bij aanmaak van de super-relaties de datatypes van alle attributen als `varchar` te definiëren en om geen verdere beperkingen (bv. primaire sleutels,...) op deze basisrelaties te leggen.

2. Importeer alle data vanuit de .csv-bestanden in de overeenkomstige super-relaties[1]. Hou rekening met de volgorde en de naam van de attributen, het scheidingsteken, het quote-symbool en het voorkomen van de header.

3. Kopieer de data vanuit elk van de super-relaties naar de verschillende basisrelaties die je hebt aangemaakt tijdens het fysiek ontwerp door middel van afzonderlijke `INSERT`-instructies (zie Sectie 3.2).

4. Verwijder de super-relaties.

Zoals reeds aangehaald is het dus niet noodzakelijk om alle beperkingen uit het fysiek ontwerp over te nemen in de super-relaties. Wanneer je de derde stap van bovenstaande werkwijze uitvoert, zal je zien dat data enkel gekopieerd kunnen worden naar de verschillende basisrelaties indien ze voldoen aan alle beperkingen die gedefinieerd zijn op deze basisrelaties.

Maak, voor elk van de twee .csv-bestanden, een super-relatie aan in de `wielrennen` databank. Zorg ervoor dat deze super-relaties alle attributen bevatten die voorkomen in de corresponderende .csv-bestanden. Geef deze super-relaties de naam `super_renners` en `super_uitslagen`.

Nu je twee super-relaties hebt aangemaakt, is het tijd om alle data uit de twee .csv-bestanden in de overeenkomstige super-relaties te importeren. Het inladen van data uit een extern .csv bestand kan in pgAdmin 4 door rechts te klikken op de naam van

[1] Indien je een foutmelding krijgt in verband met het 'binary path in pgAdmin', klik dan zeker eens op volgende link: `https://dba.stackexchange.com/questions/149169/binary-path-in-the-pgadmin-preferences`

de basisrelatie waarin je data wil importeren en dan ‘Import/Export Data...’ te selecteren. In het venster dat vervolgens opent, klik je ‘Import’ aan en selecteer je het gewenste .csv bestand. Zorg ervoor dat je juist aangeeft (onder ‘Options’) of het .csv bestand een header bevat en wat het scheidingsteken (delimiter) en het quote-symbool zijn.

Importeer de data uit de twee gegeven .csv-bestanden in de overeenkomstige super-relaties. Kijk zeker eens na, vooraleer verder te gaan, of de data correct en volledig geïmporteerd zijn.

De instructie[2] die pgAdmin 4 achter de schermen zal uitvoeren is van de vorm

```
COPY basisrelatie  [(attr1,...,attrn)]
    FROM ‘bestandsnaam’
    DELIMITER ‘scheidingsteken’
    QUOTE ‘quote-symbool’
    CSV HEADER;
```

Met deze SQL-instructie kan data uit het opgegeven bestand eenvoudig in de opgegeven basisrelatie geïmporteerd worden. Naar het bestand wordt gerefereerd door middel van zijn (absolute of relatieve) padnaam. De attribuutnamen die worden meegegeven aan de instructie moeten voorkomen in dezelfde volgorde dan gegeven in het .csv bestand. Indien het .csv bestand minder attributen bevat dan opgegeven in deze lijst, worden de resterende kolommen aangevuld met `NULL`-waarden of default-waarden[3]. Indien het .csv bestand minstens evenveel attributen bevat en deze attributen in het .csv bestand in de opgegeven volgorde staan, moeten de attribuutnamen niet expliciet worden opgegeven (dit wordt aangegeven door de vierkante haakjes, die geen onderdeel zijn van de syntax). Ook is het belangrijk dat de ingeladen waarden datatype-compatibel zijn. Dit betekent dat ze moeten bestaan binnen het domein van het datatype van het overeenkomstige attribuut in de opgegeven basisrelatie. Het `DELIMITER`-keyword geeft aan welk karakter er gebruikt wordt om attribuutwaarden in het .csv bestand te scheiden. In de meeste gevallen is dit een komma[4], maar dit kan ook een puntkomma, een tab... zijn. Het `QUOTE`-keyword geeft dan weer aan welk karakter er gebruikt om attribuutwaarden in het .csv bestand te quoteren. Tenslotte geeft `CSV HEADER` aan dat de eerste rij van het .csv bestand de kolomnamen bevat en dus niet geïmporteerd moet worden als data in de opgegeven basisrelatie.

[2] `https://www.postgresql.org/docs/current/sql-copy.html`

[3] Een default-waarde is een waarde die wordt toegevoegd aan een attribuut wanneer er geen waarde voor dit attribuut is gedefinieerd en kan vastgelegd worden bij de aanmaak van de basisrelaties.

[4] Hierbij nogmaals de herinnering dat ‘csv’ staat voor ‘comma-separated values’.

3.2 Data kopiëren tussen basisrelaties

Een tweede methode om data toe te voegen aan een bestaande basisrelatie is door deze data (deels) te kopiëren vanuit een andere basisrelatie. Zo kan je bijvoorbeeld data kopiëren vanuit bepaalde super-relaties naar de verschillende basisrelaties die je hebt geïmplementeerd tijdens het fysiek ontwerp. Om data op te vragen uit een basisrelatie kan je gebruik maken van eenvoudige `SELECT`-queries.

3.2.1 Opvragen

Zoals hierboven reeds werd aangegeven, kan je door middel van `SELECT`-queries[5] bepaalde data uit een basisrelatie selecteren. Het kan hier om heel complexe, analytische queries gaan, die je typisch gebruikt om inzichten te verwerven uit de beschikbare data, om berekeningen uit te voeren ter ondersteuning van bedrijfsprocessen... Het opstellen van dergelijke, meer geavanceerde queries zal later in dit vak uitgebreid aan bod komen. In de context van het importeren van data dienen we echter alleen maar `SELECT`-queries op te stellen die gebruikt kunnen worden om data te kopiëren tussen verschillende basisrelaties. In deze context volstaan meestal vrij eenvoudige `SELECT`-queries, die typisch de volgende vorm aannemen.

```
SELECT ... FROM ... WHERE ...;
```

Na het `SELECT`-keyword volgt (in de meest eenvoudige vorm) een oplijsting van kolomnamen waarvan je de onderliggende waarden wil opvragen. De `FROM`-clausule bepaalt de basisrelatie(s) waaruit je data wil opvragen. Let op dat je, na het `SELECT`-keyword, enkel kolomnamen kan oplijsten van kolommen die voorkomen in de basisrelatie(s) die je meegeeft na het `FROM`-keyword. De `WHERE`-clausule dient dan weer als filter. Deze filter zorgt ervoor dat enkel rijen die voldoen aan de booleaanse expressie die wordt opgegeven na dit `WHERE`-keyword teruggegeven zullen worden. Een eenvoudig voorbeeld van een `SELECT`-query is

```
SELECT
    wedstrijd_naam,
    rit_nr,
    datum,
    type,
    beschrijving,
    route_id
FROM super_uitslagen
WHERE wedstrijd_naam = 'Tour de France';
```

Deze instructie geeft alle data (ritnummer, datum, type...) terug met betrekking tot ritten die plaatsvonden in de wedstrijd met naam 'Tour de France'.

[5] `https://www.postgresql.org/docs/current/sql-select.html`

Voer bovenstaande instructie uit in de query tool van pgAdmin 4 en interpreteer het resultaat.

Het uitvoeren van deze query resulteert in een tabel met de gewenste data en attributen. Daarnaast zou het je moeten opvallen dat sommige rijen (ritten) meermaals voorkomen in deze tabel. Sommige ritten corresponderen immers met meerdere rijen in het `super_uitslagen.csv` bestand, omdat er meerdere uitslagen gelinkt kunnen zijn aan een bepaalde rit (namelijk één voor elke wielrenner die deelgenomen heeft aan deze rit). Om te vermijden dat eenzelfde rij meerdere keren voorkomt in het resultaat van een `SELECT`-query, kan je het `DISTINCT`-keyword toevoegen aan de `SELECT`-clausule.

Voer onderstaande instructie uit in de query tool van pgAdmin 4 en interpreteer het resultaat opnieuw.

```
SELECT DISTINCT
    wedstrijd_naam,
    rit_nr,
    datum,
    type,
    beschrijving,
    route_id
FROM super_uitslagen
WHERE wedstrijd_naam = 'Tour de France';
```

Het verschil met de eerste query is dat bij uitvoering van bovenstaande query elke rij in de resultatentabel slechts een keer wordt getoond. Je kan dus vaststellen dat er in de gegeven dataset 21 unieke ritten zijn opgeslagen die plaatsvonden in de wedstrijd met naam 'Tour de France'. In de context van het kopiëren van data tussen basisrelaties is het `DISTINCT`-keyword vrij belangrijk. We willen immers dat er enkel unieke combinaties van waarden voor de attributen `wedstrijdnaam` en `nr` in de tabel `rit` worden opgeslagen, aangezien dit de primaire sleutel is van deze tabel. Zonder toevoeging van dit `DISTINCT`-keyword hebben we hiervoor geen garantie (zie Sectie 3.2.2).
Indien je de data met betrekking tot alle ritten wil selecteren (en niet enkel met betrekking tot ritten die plaatsvonden in de wedstrijd met naam 'Tour de France'), kan je de `WHERE`-clausule in zijn geheel weglaten.

Selecteer de data met betrekking tot alle ritten die opgeslagen zijn in `super_uitslagen`. Zorg ervoor dat elke rij in de resultatentabel slechts een keer wordt getoond. Bekijk en interpreteer de resultatentabel aandachtig. Hoeveel unieke ritten zijn er opgeslagen in de gegeven dataset?

Een volgende functionaliteit van SQL die nuttig kan zijn in de context van het kopiëren van data is het omzetten van waarden van een bepaald datatype naar een ander datatype. In de super-relaties die je hebt aangemaakt worden alle data als tekst (datatype `varchar`) opgeslagen. De basisrelaties die je hebt aangemaakt tijdens het fysiek ontwerp bestaan echter uit verschillende attributen die een ander datatype hebben (bijvoorbeeld `integer`, `date`...). Gelukkig voorziet SQL de mogelijkheid om eenvoudig attribuutwaarden om te zetten naar een ander datatype ('casten'). Door middel van onderstaande `SELECT`-query worden de waarden uit de geselecteerde kolommen omgezet comform met de gedefinieerde datatypes in de tabel `rit`.

```
SELECT DISTINCT
    wedstrijd_naam,
    rit_nr::integer,
    datum::date,
    type,
    beschrijving,
    route_id::integer
FROM super_uitslagen;
```

De generieke syntax om een attribuut met naam `attribuutnaam` om te zetten naar het datatype `datatype` is `attribuutnaam::datatype`. Let wel op dat dit enkel mogelijk is indien alle waarden die het attribuut aanneemt in de opgegeven basisrelatie effectief omgezet kunnen worden naar het andere datatype. Het zal bijvoorbeeld niet lukken om de waarden die voorkomen onder het attribuut `wedstrijd_naam` te casten naar het datatype `integer`.
Tot slot is het mogelijk om de resultaten van meerdere `SELECT`-queries te combineren, omdat er bijvoorbeeld bepaalde data in meerdere tabellen en/of kolommen voorkomen. Dit kan je doen door gebruik te maken van een query van de vorm

```
select-query-1
    UNION
select-query-2;
```

Deze query combineert dus de resultaten van beide 'sub'-queries `select-query-1` en `select-query-2` (dit kunnen er ook meer dan twee zijn) en houdt enkel unieke

rijen over. Let wel op, het aantal kolommen dat als resultaat wordt teruggegeven door `select-query-1` en `select-query-2` dient gelijk te zijn. Bovendien moeten de datatypes van de kolommen die teruggegeven worden door `select-query-1` en `select-query-2` compatibel zijn met elkaar. In onderstaande opdracht gaan we op zoek naar alle unieke namen van renners die voorkomen in `super_renners` of `super_uitslagen`.

1. Selecteer alle waarden die voorkomen onder het attribuut `renner_naam` uit `super_renners`.
2. Selecteer alle waarden die voorkomen onder het attribuut `renner_naam` uit `super_uitslagen`.
3. Combineer de resultaten van bovenstaande queries door middel van de `UNION` operator en interpreteer aandachtig het resultaat.

3.2.2 Kopiëren

Je hebt net geleerd hoe je bepaalde data kan selecteren uit een (of meerdere) basisrelatie(s) door middel van (eenvoudige) `SELECT`-queries. We willen nu echter de geselecteerde data ook kunnen toevoegen aan (of, met andere woorden, kunnen kopiëren naar) een andere basisrelatie. De SQL-instructie[6] om alle data die voorkomen in het resultaat van een `SELECT`-query toe te voegen aan een basisrelatie is van de vorm

```
INSERT INTO basisrelatie  [(attr1,...,attrn)] select-query;
```

Merk op dat '`select-query`' ingevuld kan worden door eender welke `SELECT`-query (bv. de query die hierboven gebruikt werd om de data met betrekking tot alle unieke ritten op te vragen). Let wel op, het resultaat van deze query moet opnieuw (minstens) evenveel attributen bevatten als het aantal in te vullen attributen in de opgegeven basisrelatie. Daarnaast moeten de waarden opnieuw type-compatibel zijn, speelt de volgorde van voorkomen een rol én moeten alle geselecteerde data voldoen aan de beperkingen die gedefinieerd zijn op de basisrelatie waarnaar de data worden gekopieerd.
Laat ons nu proberen om alle data met betrekking tot ritten te kopiëren vanuit de super-relatie `super_uitslagen` naar de basisrelatie `rit`. Een eerste instructie om dit te proberen is de volgende.

[6] `https://www.postgresql.org/docs/current/sql-insert.html`

```
INSERT INTO rit
    (wedstrijdnaam, nr, datum, type, beschrijving, routeid)
SELECT DISTINCT
    wedstrijd_naam,
    rit_nr::integer,
    datum::date,
    type,
    beschrijving,
    route_id::integer
FROM super_uitslagen;
```

Het is hierbij belangrijk om de aanwezigheid van het `DISTINCT`-keyword op te merken. Indien `DISTINCT` niet toegevoegd zou worden, zou de `INSERT`-instructie falen. De reden hiervoor is dat er dan meerdere rijen met dezelfde combinatie van waarden voor de attributen `wedstrijdnaam` en `nr` toegevoegd zouden worden aan de basisrelatie `wedstrijd`. Dit is echter niet toegestaan, aangezien {`wedstrijdnaam`,`nr`} de primaire sleutel vormt van deze basisrelatie en er dus unieke combinaties van waarden verwacht worden voor deze attributen. Verder is het belangrijk dat de waarden van alle attributen gecast worden naar het juiste datatype.
Wanneer je bovenstaande query probeert uit te voeren, zal je merken dat PostgreSQL toch nog steeds een foutmelding opgooit. Deze foutmelding geeft aan dat er waarden zijn die je probeert toe te voegen onder het attribuut `routeid` (resp. `wedstrijdnaam`) van de `rit` tabel die niet onder het attribuut `id` van de `route` tabel (resp. `naam` van de `wedstrijd` tabel) opgeslagen zijn. Dit is inderdaad een inbreuk op de vreemde sleutel-beperkingen[7] die gedefinieerd zijn tussen `rit` en `route`, en tussen `rit` en `wedstrijd`. Daarom is het noodzakelijk om eerst de meegeleverde data correct en volledig toe te voegen aan de tabellen `route` en `wedstrijd`, en dan pas aan de tabel `rit`. Over het algemeen is het zeer belangrijk om bij het manipuleren van data rekening te houden met de referenties tussen tabellen.

Kopieer alle data vanuit `super_renners` en `super_uitslagen` naar de basisrelaties die je hebt aangemaakt tijdens het fysiek ontwerp. Hou rekening met alle beperkingen die gedefinieerd zijn op deze basisrelaties.

Nadat je alle data hebt gekopieerd naar de verschillende basisrelaties, kan je aan de hand van Tabel 1 controleren of het aantal rijen in een door jou aangemaakte basisrelatie overeenkomt met het verwachte aantal rijen. Dit kan je doen door voor iedere basisrelatie het aantal rijen te tellen met een instructie van de vorm

[7] Denk nog eens na over wat een vreemde sleutel-beperking juist afdwingt.

```
SELECT COUNT(*) FROM basisrelatie;
```

waarin je ‘basisrelatie’ vervangt door de naam van de basisrelatie waarvan je het aantal rijen wil tellen.

basisrelatie	**verwachte aantal rijen**
wielerteam	32
wielrenner	934
route	93
wedstrijd	13
rit	93
uitslag	14731

Tabel 1: Verwacht aantal rijen per basisrelatie.

Verwijder de super-relaties `super_renners` en `super_uitslagen`.

3.3 Expliciet waarden toevoegen

Een laatste methode om data toe te voegen aan een bestaande basisrelatie is door expliciet waarden voor een of meerdere rijen op te geven en deze rijen toe te voegen aan de basisrelatie. Dit kan opnieuw door middel van een `INSERT`-instructie, meerbepaald door een instructie van de vorm

```
INSERT INTO basisrelatie  [(attr1,...,attrn)]
    VALUES (waarde1,...,waarden);
```

Ook voor deze instructie gelden dezelfde regels als voor de instructies die we geïntroduceerd hebben in Sectie 3.1 en Sectie 3.2. Daarnaast is het mogelijk om, mits een kleine aanpassing aan deze instructie, waarden voor meerdere rijen binnen eenzelfde op te geven.

Voeg data toe conform met het volgende scenario. Op 4 maart 2023 vond de 17e editie van de Italiaanse eendagswedstrijd ‘Strade-Bianche’ plaats. Deze wedstrijd startte en eindigde in ‘Siena’. De route had een lengte van 184 kilometer, omvatte 3107 hoogtemeters, en kreeg moeilijkheid 3 toegekend. De wedstrijd werd, na een korte solo, gewonnen door ‘Thomas Pidcock’. Hij finishte na 4 uur, 31 minuten en 41 seconden.

4 Exporteren van data

Tenslotte willen we jullie nog even laten kennismaken met het exporteren van data uit een basisrelatie naar een .csv bestand. Dit kan je in pgAdmin 4 doen op een manier die gelijkaardig is aan het importeren van data (zie Sectie 3.1), namelijk door rechts te klikken op de basisrelatie waarvan je de data wil exporteren en 'Import/-Export Data...' te selecteren. Je zorgt dat de optie 'Export' is geselecteerd, geeft de padnaam van het bestand waar je de data naar wil wegschrijven op en kiest als formaat 'csv'. Tot slot geef je aan of je de kolomnamen op de eerste rij wil toevoegen met de 'Header' optie en kies je een scheidingsteken met de 'Delimiter' optie.

Exporteer de data uit de basisrelatie `wielrenner` naar een .csv bestand. Bekijk aandachtig de inhoud van dit bestand.

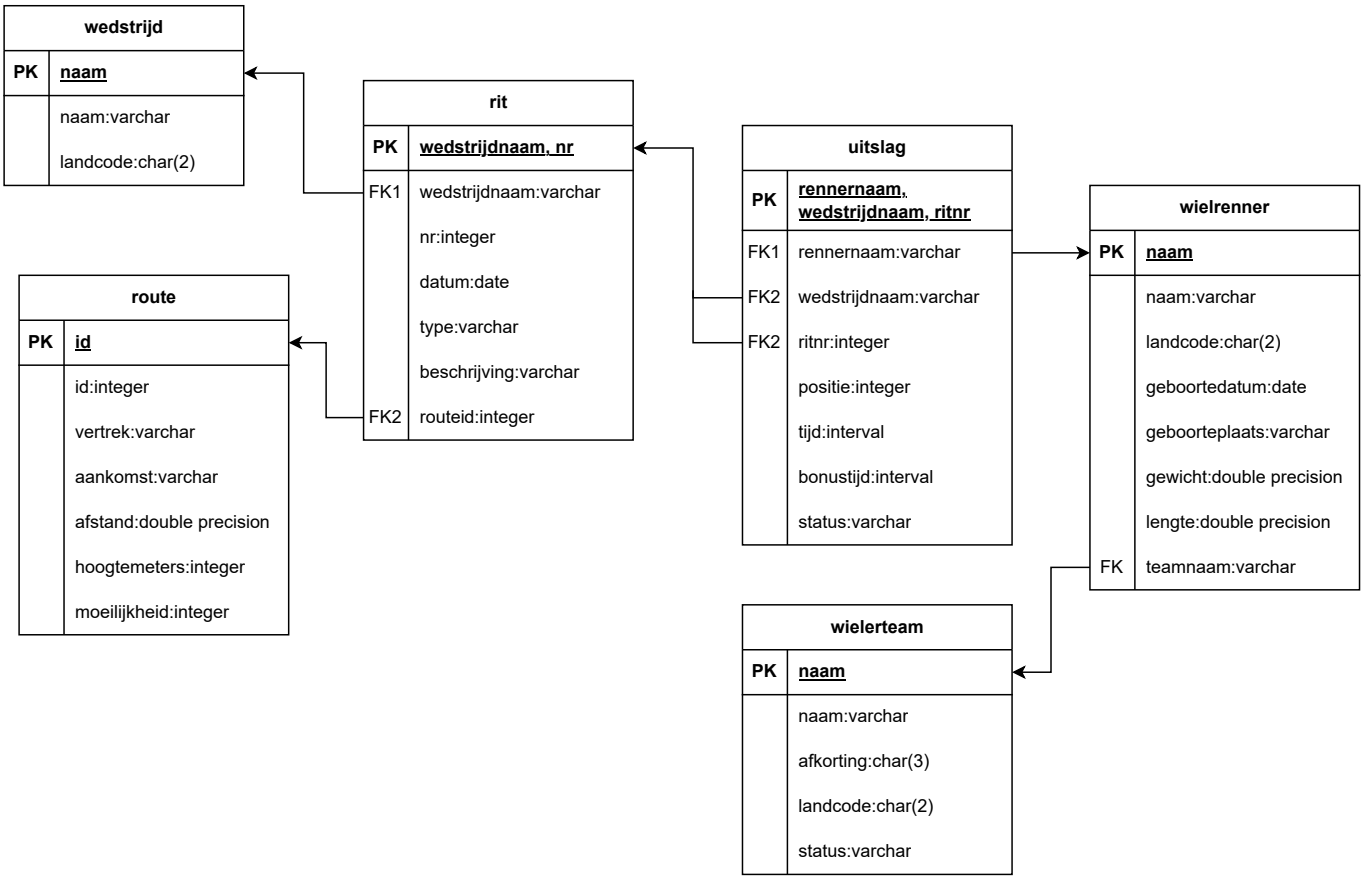
5 Backup

In de workshop 'Introductie PostgreSQL' werd er getoond hoe je een backup kan maken van een PostgreSQL databank door middel van het `pg_dump` commando. Aangezien er nu data toegevoegd zijn aan de `wielrennen` databank, willen we echter niet enkel een backup maken van het databankschema, maar ook van de data. Dit kan door uitvoering van het onderstaande commando op de commandolijn.

```
pg_dump
    --host=127.0.0.1
    --port=5432
    --dbname=wielrennen
    --username=postgres
    --clean
    --create
    --if-exists
    --file=wielrennen.sql
```

Appendix: Relationeel schema `wielrennen` databank

In onderstaande figuur vind je het relationeel databankschema van de `wielrennen` databank. Hierbij wordt iedere basisrelatie weergegeven door een rechthoek, die bovendien een oplijsting van alle attributen met bijhorende datatypes bevat. Daarnaast worden de attributen die behoren tot de primaire sleutel (PK) bovenaan weergegeven, en worden vreemde sleutels (FKx) voorgesteld door een pijl tussen de betreffende attribuutverzamelingen. Alle extra beperkingen die niet kunnen worden weergegeven in dit schema, worden hieronder opgelijst.



Extra beperkingen

- `wielerteam`:
  - uniek: {afkorting}
  - check: status ∈ {‘World Tour’, ‘Pro Tour’}
- `wielrenner`:
  - optioneel: geboorteplaats, gewicht, lengte
  - check: gewicht > 0
  - check: lengte > 0
- `route`:
  - check: afstand > 0
  - check: hoogtemeters ≥ 0
  - check: moeilijkheid ∈ [1,5]
- `rit`:
  - check: nr ≥ 1
  - check: type ∈ {‘Road Race’, ‘Individual Time Trial’, ‘Team Time Trial’}
- `uitslag`:
  - optioneel: positie, tijd, bonustijd
  - check: positie > 0
  - check: tijd > 0
  - check: bonustijd ≥ 0
  - check: status ∈ {‘Did Finish’, ‘Did Not Finish’,‘Did Not Start’, ‘Over Time Limit’, ‘Disqualified’}
  - check: status = ‘Did Finish’ ⇒ positie is not null ∧ tijd is not null ∧ bonustijd is not null
  - check: status ≠ ‘Did Finish’ ⇒ positie is null
  - check: status ∈ {‘Did Not Finish’, ‘Did Not Start’} ⇒ tijd is null
  - check: status = ‘Did Not Start’ ⇒ bonustijd is null
  - trigger: een wielrenner kan op eenzelfde datum slechts aan 1 rit deelnemen
  - trigger: indien de uitslagstatus van een renner in een rit niet gelijk is aan ‘Did Finish’, kan hij/zij niet meer deelnemen aan ritten in dezelfde wedstrijd met een hoger volgnummer